```
/* Program9: Design, Develop and Implement a menu driven Program in C for the following
operations on Binary Search Tree (BST) of Integers
a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (KEY) and report the appropriate message
d. Exit. */

#include<stdio.h>
#include<stdlib.h>
struct BST
{
	int data;
	struct BST *lchild;
	struct BST *rchild;
};
typedef struct BST * NODE;

NODE create()
{
	NODE temp;
	temp = (NODE) malloc(sizeof(struct BST));
	printf("\nEnter The value: ");
	scanf("%d", &temp->data);


	temp->lchild = NULL;
	temp->rchild = NULL;
	return temp;
}

void insert(NODE root, NODE newnode);
void inorder(NODE root);
void preorder(NODE root);
void postorder(NODE root);
void search(NODE root);

void insert(NODE root, NODE newnode)
{
   /*Note: if newnode->data ==  root->data it will be skipped. No duplicate nodes are allowed */

	if (newnode->data < root->data)
	{
		if (root->lchild == NULL)
			root->lchild = newnode;
		else
			insert(root->lchild, newnode);
	}
	if (newnode->data > root->data)
	{
		if (root->rchild == NULL)
			root->rchild = newnode;
		else
			insert(root->rchild, newnode);
	}
}


void search(NODE root)
{
	int key;
	NODE cur;
	if(root == NULL)
	{
		printf("\nBST is empty.");
		return;
	}
	printf("\nEnter Element to be searched: ");
	scanf("%d", &key);
	cur = root;
	while (cur != NULL)
	{
		if (cur->data == key)
		{
			printf("\nKey element is present in BST");
			return;
		}
		if (key < cur->data)
			cur = cur->lchild;
		else
			cur = cur->rchild;
	}
	printf("\nKey element is not found in the BST");
}

void inorder(NODE root)
{
	if(root != NULL)
	{
		inorder(root->lchild);
		printf("%d ", root->data);
		inorder(root->rchild);
	}
}

void preorder(NODE root)
{
        if (root != NULL)
        {
                printf("%d ", root->data);
                preorder(root->lchild);
                preorder(root->rchild);
        }
}

void postorder(NODE root)
{
        if (root != NULL)
        {
                postorder(root->lchild);
                postorder(root->rchild);
                printf("%d ", root->data);
        }
}


void main()
{
        int ch, key, val, i, n;
        NODE root = NULL, newnode;
        while(1)
        {
                printf("\n~~~~BST MENU~~~~");
                printf("\n1.Create a BST");
                printf("\n2.BST Traversals: ");
                printf("\n3.Search");
                printf("\n4.Exit");
                printf("\nEnter your choice: ");
                scanf("%d", &ch);
                switch(ch)
                {
                        case 1:          printf("\nEnter the number of elements: ");
                                        scanf("%d", &n);
                                        for(i=1;i<=n;i++)
                                        {
                                                newnode = create();
                                                if (root == NULL)
                                                        root = newnode;
                                                else
                                                        insert(root, newnode);
                                        }
                                        break;
                         case 2:          if (root == NULL)
                                                printf("\nTree Is Not Created");
                                        else
```

```
                                        {
                                                printf("\nThe Preorder display : ");
                                                preorder(root);
                                                printf("\nThe Inorder display : ");
                                                inorder(root);
                                                printf("\nThe Postorder display : ");
                                                postorder(root);
                                        }

                                        break;
                        case 3:          search(root);
                                        break;

                        case 4:     exit(0);
                }
        }
}
```